REMARKS

Claims 1, 3, 4, 6-10, and 14 have been amended to further clarify the present invention without raising new issues.

The present invention relates from the discovery that by taking processing elements and forming them into one or more groups such that one group receives one instruction so that the instructions are executed in parallel, the utilization efficiency of hardware resources is improved and the time required for multimedia data processing are reduced, even when a memory stores pieces of data that requires different types of operation.

Examiner has rejected Claims 1 and 14 under 35 U.S.C. §103(a) as being unpatentable over *Knight, Jr.* (U.S. Patent No. 4,825,360, hereinafter "*Knight*") in view of *Emma et al* (U.S. Patent No. 5,297,281, hereinafter "*Emma*").

As the Examiner is aware, the hard question is whether the hypothetical combination is based upon hindsight from the present teaching rather than what would be obvious apart from the present teaching to a person of ordinary skill in this field.

Claim 1 and *Knight*

Knight seeks to solve the problem of intermixing strictly functional and object oriented programming languages by allowing the computer system to execute the program in parallel while giving the appearance to the programmer that the program is being executed sequentially. (Col. 3, line 65 – Col. 4 line 2). The major hurdle is side-effects, which seek to alter the main memory, which make the construction of parallel architectures difficult. (Col. 2, lines 20-25). Thus, *Knight* teaches eliminating unnecessary side effects by confirming in order and enforcing the dependencies of later blocks on earlier side effects. (Col. 4, lines 50-60). *Knight* requires that the instructions be predominantly function and when broken down into blocks, each block

terminate in only one side-effect. (Col. 4, lines 38-49; Col. 5, lines 63-70). As preparation, a compiler transforms the program into a sequence of compiled primitive instructions called transaction bocks. Each transaction block has a mostly functional portion and terminates in only a side effect. (Col. 4, lines 38-49; Col. 5, lines 63-70) Then, the blocks are executed independently on several processors.

As can be seen in Figure 1, each processor 1a-1c fetches a block and they begin executing the blocks in parallel. During execution, each reference to the main memory is stored in dependency cache 3a-3c whereas side effecting instructions are stored in confirm caches 2a-2c. References to the memory do not change the memory, but rather retrieves data from the memory whereas side-effecting instructions are supposed to alter the main memory after confirmation. By storing them in caches, this prevents the main memory 6-7 from being altered prior to confirmation. Thus, each block must be confirmed and the order of confirmation is crucial. To achieve this, block counter 8 is utilized, which contains the order for confirming the blocks.

Assuming processor 1b is supposed to be confirmed first, after the confirmation of processor 1b, the data stored in confirmed cache 2b is sent to processors 1a and 1c. Now, processor 1a and 1c compare the confirmed cache 2b to the data stored in dependency cache 3a and 3c. If 3a and 3c reference a memory location which was modified by confirmed cache 2b, then the results of processor 1a and 1c are now incorrect. The results must be discarded and processor must begin execution of the block again with the updated information in confirmed cache 2b. (Col. 4 line 25 – Col. 5 line 35; Col. 7 line 28 – Col. 8 line 32).

The present invention can dynamically group the processors together by the number of instructions to be executed. This is accomplished by using grouping units such as the grouping

unit 250 in Figure 5 or grouping unit 350 in Figure 9. Thus, if there are 128 processing units, for example, and 128 instructions, each individual processing unit can be a group. However, if there are only 64 instructions, two processing units would form a group with each group executing a piece of instruction meaning there are two processing units that would divide up the work of processing a piece of instruction. Likewise, if there are only 32 instructions, four processing units would form a group with each group executing a piece of instruction meaning four processing units work in conjunction with each other to process the piece of instruction.

In contrast, *Knight* does not teach the feature of "a group forming unit operable to form the processing elements into as many groups as the number of instructions included in the instruction sequence" like the grouping unit 250. (Figures 6, 13). In the present invention, the grouping unit 250 receives a grouping information from the instruction decoding unit 420 via the grouping information obtaining unit 253. (Specification pg. 50, lines 18-21; Figures 6, 13). The grouping information is included in a piece of instruction data that is fetched by the instruction fetching unit 110 which is obtained by the instruction decoding unit 420 and passed on to the grouping unit 250. (Specification pg. 50, lines 21-23). Figures 14A to 14E show the data structures of sample pieces of instructions with the grouping information in them. After obtaining the grouping information, the grouping unit looks at the combination storing unit 252 to determine how the processing elements should be grouped. (Specification pg. 37, lines 19-22; Figures 6, 7, 13). The combination storing unit 252 is a Read Only Memory that stores a table of all the various combinations of grouping for the processor elements indexed by grouping information, as can be seen in Figure 7. (Specification pg. 37, line 19 – pg. 38, line 14). After retrieving the proper group formation from the combination storing unit 252, the grouping unit 250 then sends the appropriate signals out to the controlling signal outputting unit 255a-d so that

it reaches the appropriate processing elements 130a-d based on which groups the processing elements belong to. (Specifications pg. 38, line 19 – pg. 39, line 12).

In *Knight*, however, it is the instructions rather than the processing elements that are broken down into blocks such that the blocks terminate in only a single side-effect. (Col. 4, lines 38-49; Col. 5, lines 63-70; Claim 1). The compiler in *Knight* is not concerned with the formation of the processors, but rather with the formation of the instruction blocks. The compiler assumes that there will be processors that will somehow handle the instructions that it packages into a block which terminates in only one side-effect. This is highlighted by the fact that *Knight* does not utilize grouping numbers which are embedded in instructions or contain a table with the different grouping combinations possible for the processing elements. Therefore, *Knight* does not teach nor suggest dynamically group the processing elements such that a single group handles a single instruction.

Also, in *Knight*, each processing unit processes an instruction, and there is no way to utilize more than one processor to work on an instruction. So if there are 64 instructions and 128 processing units, the first 64 processing units would be working on an instruction and the second 64 processing units would sit idle or the second 64 processing unit would do the exact same work that the first 64 processing units are doing. That is, the second 64 processing units would do from the beginning the exact same instructions that the first 64 processing units are doing. This problem is magnified where there are only 32 pieces of instruction and there are 128 processing units. In that case, 32 pieces of instruction would be done by the first 32 processing units, while the second, third, and fourth 32 processing units sit idle or do duplicative work.

In addition, in *Knight*, each processor must fetch an instruction, even if each processor is working on the exact same block. However, in the present invention, if the processors are

working on the same instruction, they would be in the same group and they would receive the instructions at once. Thus, the number of instruction fetches are reduced which improves the efficiency of the system.

Furthermore, *Knight* requires that the instructions be predominantly functional language and when broken into blocks, the blocks terminate in only a single side-effect. (Col. 4, lines 38-49; Col. 5, lines 63-70; Claim 1). The present invention, however, has no limitation on the number of side-effects or the level of functional instructions in the instructions.

Thus, Claim 1 has novelty and inventiveness over *Knight*.

Claim 1 and *Emma*

Emma teaches a technique for utilizing parallel processing and also exploiting the property of persistence of behavior to execute instructions at a higher rate. (Col. 3 lines 39-46). This is accomplished by providing multiple pipeline processing stage, dividing a sequence of instructions into two groups according to a delimiting rule, storing the information identifying the groups, and channeling the information defining the different groups to different ones of the multiple pipeline processing stages for concurrent execution of the instructions which constitute the groups. (Col. 3, lines 46-line 59). In practice, instructions are divided into groups, such as G1, G2, G3, etc. under the control of the BHT 110. The BHT 110 then sends the group information to cache 112 which funnels this to instruction buffer 116. The instruction buffer 116 then sends the information to the main pipeline 124 and the auxiliary pipeline 126, which are the two processors which execute instructions. Both receive groups at the same time and both execute the groups at the same time. Fetch checks 132 and Register checks 134 then checks to see if the auxiliary pipeline 126 is producing valid data e.g. to ensure it is not using values which have been subsequently modified by main pipeline 124. If the results are valid, then they are

both stored into memory 114. If the auxiliary pipeline 126 produces an invalid result, its result is discarded and the group is now loaded onto the main pipeline 124 for execution while the auxiliary pipeline 126 receives a new group to execute. (Col. 4, line 56 – Col. 6 line 65; Figure 1). Thus, if groups G1, G2, and G3 were waiting to be executed, G1 would go to main pipeline 124 for execution while G2 would go to auxiliary pipeline 126 for execution. If the result of processing G2 is valid, the result would be stored in memory 114. If the result is invalid, the result would be discarded and G2 would now be loaded onto main pipeline 124 while the next group waiting to be executed, G3, would be loaded onto auxiliary pipeline 126.

In contrast, the present invention can utilize more than two processors. Instead of only a main pipeline and an auxiliary pipeline, the present invention can utilize a number of processor elements. (Specification pg. 22, line 9-12; Figure 1). This is significant because the more processing elements there are, the greater the ability to process multiple instructions.

In addition, *Emma* does not teach nor suggest the use of dividing the processing elements into groups. *Emma* only teaches dividing the instructions into at least two groups and then each of the two processors processes a group. In contrast, the present invention does not divide the instruction, but instead divides the processing elements into as many groups as the number of instructions included in the instruction sequence.

Also, Examiner cites *Emma* as teaching the use of a group number. However, the group number in *Emma* is related to the group number for instructions. (Col. 4, lines 38-48; Col. 5, lines 35-46). However, in the present invention, the group numbers are used to determine which group combination to use to group the processing elements such that one group executes one instruction. The group numbers reference an indexed Read Only Memory which contains all of the possible group combinations for the processing elements. Thus, while both may have group

numbers, *Emma's* group numbers are not the same nor equivalent to the group numbers in the present invention.

Thus, Claim 1 has novelty and inventiveness over Emma.

Claim 1 in view of *Knight* and *Emma*

As set forth *In re Kahn*, 441 F.3d 977, 987-88 (Fed. Cir. 2006):

> The motivation-suggestion-teaching test picks up where the analogous art test leaves off and informs the *Graham* analysis. [*Graham v. John Deere Co.*, 383 U.S. 1, 13-14 (1966).]
>
> To reach a non-hindsight driven conclusion as to whether a person having ordinary skill in the art at the time of the invention would have viewed the subject matter as a whole to have been obvious in view of multiple references, the Board must provide some rationale, articulation, or reasoned basis to explain why the conclusion of obviousness is correct. The requirement of such an explanation is consistent with governing obviousness law . . . .
>
> * * *
>
> A suggestion, teaching, or motivation to combine the relevant prior art teachings does not have to be found explicitly in the prior art, as "the teaching, motivation, or suggestion may be implicit from the prior art as the whole, rather than expressly stated in the references . . . . The test for an implicit showing is what the combined teachings, knowledge of one of ordinary skill in the art, and the nature of the problem to be solved as a whole would have suggested to those ordinary skill in the art." However, rejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness. This requirement is as much rooted in the Administrative Procedure Act [for review of Board determinations], which ensures due process and non-arbitrary decisionmaking, as it is in §103.

As can be appreciated, the more that the cited references must be modified to meet the outstanding claims, the more likely that an unintended issue of hindsight may drive the rejection. This is particularly true for an Examiner who is attempting to provide a diligent effort to ensure that only patentable subject matter occurs. The difficult issue is to step back from the zeal of the

examination process and to appreciate that the Patent Examiner has to wear both hats of advocating a position relative to the prior art, while at the same time objectively rendering in a judge-like manner, a decision on the patentability of the present claims.

An inventor seeking to combine function and non-functional language by reducing side-effects as in Knight would not look to a teaching of exploiting persistence by breaking instructions into a series of sub-instructions due to a delimiting rule for inspiration. Therefore, there is no motivation to combine *Knight* and *Emma*.

Furthermore, even if *Knight* and *Emma* were combined, however improperly, the combined references would still not teach nor suggest the features of the present invention. The resulting combination would utilize only two processors breaking predominantly functional instructions terminating in a single side-effect into sub-instructions by a delimiting rule and limiting the side-effects. It would not be capable of utilizing more than two processors. It also would not dynamically group the processing elements such that there is one group per instruction to be executed. In addition, the group number utilized in the combination would be for the instructions instead of for the processing elements. Furthermore, the instruction set would be curtailed by the requirement that the instructions be predominantly functional and when broken into blocks, the blocks terminating in only a single side-effect.

Thus, Claim 1 has novelty and inventiveness over *Knight* in view of *Emma* in view of *In re Kahn, supra.*

Independent Claim 14 and Dependent Claims.

Since Claim 14 teaches a method utilizing the features of Claim 1, it also has novelty and inventiveness over *Knight* and *Emma*. Furthermore, since Claims 1 and 14 have novelty and inventiveness over *Knight* and *Emma*, the dependent claims 3-10 which flow from the

independent claims also have novelty and inventiveness over *Knight* and *Emma* and provides additional grounds for patentability.

Conclusion

It is believed that the application is in condition for allowance and an early notification is requested.

If the Examiner believes a telephone interview will assist in the prosecution of this application, the undersigned attorney can be contacted at the listed phone number.

I hereby certify that this correspondence is being transmitted via facsimile to the USPTO at **571-273-8300** on October 19, 2006.

By: Candy Neu

Signature

Dated: October 19, 2006

Very truly yours,

SNELL & WILMER L.L.P.

Joseph W. Price
Registration No. 25,124
600 Anton Boulevard, Suite 1400
Costa Mesa, California 92626-7689
Telephone: (714) 427-7420
Facsimile: (714) 427-7799